

On the performance of different mutation operators of a subpopulation-based genetic algorithm for multi-robot task allocation problems

Chun Liu^{a,b,*}, Andreas Kroll^b

^a*School of Automation, Beijing University of Posts and Telecommunications
No 10, Xitucheng Road, 100876, Beijing, China*

^b*Department of Measurement and Control, Mechanical Engineering, University of Kassel
Mönchebergstraße 7, 34125, Kassel, Germany*

Abstract

The performance of different mutation operators is usually evaluated in conjunction with specific parameter settings of genetic algorithms and target problems. Most studies focus on the classical genetic algorithm with different parameters or on solving unconstrained combinatorial optimization problems such as the traveling salesman problems. In this paper, a subpopulation-based genetic algorithm that uses only mutation and selection is developed to solve multi-robot task allocation problems. The target problems are constrained combinatorial optimization problems, and are more complex if cooperative tasks are involved as these introduce additional spatial and temporal constraints. The proposed genetic algorithm can obtain better solutions than classical genetic algorithms with tournament selection and partially mapped crossover. The performance of different mutation operators in solving problems without/with cooperative tasks is evaluated. The results imply that inversion mutation performs better than others when solving problems without cooperative tasks, and the swap-inversion combination performs better than others when solving problems with cooperative tasks.

Keywords: constrained combinatorial optimization, genetic algorithm,

*Corresponding author.

Email addresses: `chun.liu@bupt.edu.cn` (Chun Liu),
`andreas.kroll@mrt.uni-kassel.de` (Andreas Kroll)

1. Introduction

A genetic algorithm (GA) is a centralized heuristic method inspired from biological evolution. It is widely used for optimization and search problems because of its simplicity, high flexibility in problem modeling, and good global search capability. Many genetic algorithms have been developed to solve optimization problems in bioinformatics, computational science, engineering, economics, and other fields. For example, in engineering applications, genetic algorithms have been used to solve the design of roof structures (Kociecki and Adeli, 2014), assembly problems (Akpinar and Bayhan, 2011), and industrial plant inspection (Liu and Kroll, 2012a).

Selection, crossover, and mutation operators maintain the population diversity (Mc Ginley et al., 2011), and also influence the performance of genetic algorithms. Therefore, many efforts have been devoted to the design of these operators, for example, a new selection strategy based on population recombination and elitist refinement (Kwak and Lee, 2011), a two-part chromosome crossover operator (Yuan et al., 2013), and a greedy sub tour mutation operator (Albayrak and Allahverdi, 2011) have been developed to improve the efficiency of genetic algorithms. Crossover and mutation are the main search operators of genetic algorithms. They play different roles in genetic algorithms: crossover tends to preserve the features of the parents, while mutation tends to make some small local perturbation of individuals. Compared to crossover, mutation is usually considered as a secondary operator with small probability in classical genetic algorithms (Holland, 1992). This could be due to the fact that a large mutation rate would make genetic algorithms to search randomly. However, there has not been any theoretical proof that crossover has general advantages over mutation (Spears, 1992). Many studies have shown that genetic algorithms

Symbol	Explanation
A	Solution of the multi-robot task allocation problem
A_k	Task assignment and schedule of robot R_k
$C_k(A_k)$	Time for R_k to complete its tasks A_k
c_{ijk}^t	Traveling time of robot R_k from inspection position of subtask P_i to that of P_j
c_{jk}^w	Waiting time of robot R_k to execute subtask P_j after arriving at the inspection position of P_j
eli_cnt	Elite count
gen_num	Number of generations
J	Cost function (completion time)
J_{\max}	Maximum completion time
J_{mean}	Mean completion time
J_{\min}	Minimum completion time
K	Number of subpopulations
N^P	Number of subtasks
N^R	Number of robots
N^T	Number of tasks
p_a	Probability of producing a new gene-apportion
p_c	Crossover probability
p_m	Mutation probability
pop_siz	Population size
pop_sub	Subpopulation size
P	Set of subtasks
P_i	i -th subtask
R	Set of robots
R_k	k -th robot
T	Set of tasks
T_l	l -th task
tor_siz	Tournament size
μ	Mean parameter of normal distribution for producing new gene-apportions
σ	Standard deviation of normal distribution for producing new gene-apportions
τ_i^a	Arrival time of robot at inspection position of P_i

without crossover can perform better than classical genetic algorithms, if mutation is combined with an effective selection operator (Fogel and Atmar, 1990;

Walkenhorst and Bertram, 2011; Liu and Kroll, 2012b; Osaba et al., 2014).

Mutation is usually carried out with a single parent and plays an important role in increasing the population diversity. Various mutation operators have been developed for different solution representations, for example, Gaussian and uniform mutation for binary coding (Fogel and Atmar, 1990), swap and insertion for integer coding (Larrañaga et al., 1999), polynomial and power mutation for real coding (Deep and Thakur, 2007; Deb and Deb, 2012). Some mutation operators are problem-dependent, such as greedy sub tour mutation for traveling salesman problems (Albayrak and Allahverdi, 2011) and energy mutation for multicast routing problems (Karthikeyan et al., 2013). Some studies suggest a mutation-combination (Deep and Mebrahtu, 2011) or self-adaptive mutation operators (Hong et al., 2000; Serpell and Smith, 2010; Mc Ginley et al., 2011). The performance of different mutation operators has been analyzed, showing that it highly depends on the parameter choice of genetic algorithms (Brizuela and Aceves, 2003; Wang and Zhang, 2006; Osaba et al., 2014) and the type of problems (Hasan and Saleh, 2011; Karthikeyan et al., 2013). Most of related work has studied problems without cooperative tasks such as traveling salesman problems (Deep and Mebrahtu, 2011; Albayrak and Allahverdi, 2011) and flow shop scheduling (Nearchou, 2004; Wang and Zhang, 2006). In this paper, the performance of mutation operators will be analyzed when solving multi-robot task allocation problems with cooperative tasks.

Multi-robot task allocation (MRTA) determines the task sequence and distribution for a group of robots in multi-robot systems (Gerkey and Mataric, 2004). It is a constrained combinatorial optimization problem, which usually provides solutions to minimize the cost while satisfying operational constraints. To find the global optimal solution, genetic algorithms (Liu and Kroll, 2012a) and hybrid genetic algorithms (Liu and Kroll, 2014) have been developed to solve MRTA problems without/with cooperative tasks. MRTA problems without cooperative tasks are similar to multiple traveling salesman problems. They are NP- (non-deterministic polynomial-time) hard optimization problems as traveling salesman problems are NP-hard. MRTA problems with cooperative tasks

are more complex and strongly NP-hard (Gerkey and Matarić, 2004), because each cooperative task requires at least two robots to carry it out simultaneously, which introduces both spatial and temporal constraints into the optimization problem.

In this paper, a subpopulation-based genetic algorithm is developed to solve MRTA problems. This genetic algorithm deploys mutation operators and elitism selection in each subpopulation but not any crossover operator. The effects of using different mutation operators on algorithm performance are analyzed when solving MRTA problems without/with cooperative tasks.

This paper is organized as follows: multi-robot task allocation problems with cooperative tasks are introduced in Section 2. Section 3 presents the subpopulation-based genetic algorithm. Simulation studies and the analysis of results are shown in Section 4. Finally, the conclusions are drawn in Section 5.

2. Multi-robot task allocation problems with cooperative tasks

Multi-robot task allocation is a combinatorial optimization problem, which assigns a set of tasks to a group of robots where typically the number of tasks is significantly larger than the number of robots. For solving this optimization problem, the first important thing is to understand what the tasks are. In general, the tasks can be classified into single-robot tasks and multi-robot tasks (Gerkey and Matarić, 2004). A single-robot task is carried out by a single robot. A multi-robot task requires multiple robots to perform at the same time, which is also referred to as cooperative task in this paper. Tasks vary in different practical applications. The problem complexity increases with the number of robots required for each cooperative task.

This paper studies the problem of multi-robot task allocation for industrial plant inspection using remote sensing to detect gas and fluid leakages (Bonow and Kroll, 2013; Ordoñez Müller and Kroll, 2013). The applied sensing technology requires a diffuse reflecting background for a maximum measurement range of approximate 30 m. Larger ranges can be achieved and cases without

reflecting background can be handled by using an assistant robot with a special retro-reflector (Ordoñez Müller and Kroll, 2014). This results in two types of tasks: single- and two-robot tasks. Each single-robot task is performed by one robot with an active sensor. Each two-robot task is carried out by two robots cooperatively: the first robot with an active sensor and the second robot with a retro-reflector. Multi-robot task allocation for inspection problems with cooperative tasks introduces spatial and temporal constraints: *spatial constraints*, as tasks must be executed by robots each from specific inspection position; *temporal constraints*, as each cooperative task requires two robots to carry it out at the very same time.

The objective of multi-robot task allocation problems is usually to minimize the total mission cost due to energy consumption, completion time, and/or traveled distance. Inspection problems can be safety-critical, so the inspection of the whole plant is usually required to be finished as soon as possible to avoid economic loss and environmental pollution. Quicker inspection also means that the higher frequency of inspecting a plant becomes possible or that more inspection problems can be solved in a given time. Therefore, the objective of the studied multi-robot task allocation problem in this paper is defined as the completion time. This is the time span between the first robot starting its work and the last robot finishing its tasks. Formally, given a set of robots $R = \{R_k | k \in \{1, 2, \dots, N^R\}\}$ and a set of tasks $T = \{T_l | l \in \{1, 2, \dots, N^T\}\}$, the objective (completion time) can be represented as

$$J(A) = \max_{k \in \{1, \dots, N^R\}} C_k(A_k), \quad (1)$$

where A is an admissible solution of the task allocation problem, A_k is the task sequence of robot R_k , and $C_k(A_k)$ is the time of robot R_k required to finish all assigned tasks according to the sequence A_k . The objective of multi-robot task allocation problems is to find the task allocation that minimizes $J(A)$.

Denoting each single-robot task as a subtask and each cooperative task (two-robot task) as two subtasks, all subtasks would form a set $P = \{P_i | i \in$

$\{1, 2, \dots, N^P\}$. The task allocation A must satisfy the following constraint:

$$A = \{A_k | \bigcup_{k=1}^{N^R} A_k = P, A_k \cap A_i = \emptyset\}, \quad (2)$$

with $i, k \in \{1, 2, \dots, N^R\}$. The constraint (2) ensures that each subtask is executed only once. The task allocation must also satisfy that all robots start and end their mission at their home bases; hence, the completion time (1) also involves the traveling time of each robot from its home base to its first task and the traveling time of each robot for returning from its last task to its home base.

In addition, the following three executability constraints (EC) must also be satisfied to ensure that the task allocation is feasible for execution:

- (EC1) Each cooperative task is carried out by two different robots.
- (EC2) Two subtasks of each cooperative task are started at the same time.
- (EC3) The schedule of cooperative tasks is feasible for execution, i.e., the sequence of cooperative tasks is not contradictory.

For instance, two cooperative tasks T_1 and T_2 are assigned to robots R_k and R_s . Robot R_s must carry out T_1 first if R_k perform T_1 first; otherwise, the solution is infeasible.

Based on the described characteristics and definition of multi-robot task allocation for inspection problems, it is obvious that the completion time includes the traveling time between tasks, the inspection time of each task, and the waiting time occurring when performing cooperative tasks. In this work, the traveling time is calculated using the A* algorithm for a given inspection environment. The inspection time is predefined according to the inspection method and measurement system properties. The waiting time depends on the solution itself and is calculated for each solution candidate during the execution of the genetic algorithm.

3. Subpopulation-based genetic algorithm

A subpopulation-based genetic algorithm is developed to solve the multi-robot task allocation problems with cooperative tasks in this section. At the

beginning of this section, the solution representation is introduced. After that, the implementation of the proposed genetic algorithm is illustrated. At the end, the subpopulation-based and a classical genetic algorithm (Taplin et al., 2005) are compared.

3.1. Solution representation

Permutation coding is used to represent a solution of this optimization problem, because it is the most natural and readable way to represent a task sequence. This representation can be very easily implemented in the most commonly used programming languages such as MATLAB or C/C++. Using permutation coding, a solution of multi-robot task allocation problems with cooperative tasks is composed of N^R task sequences that involve the distribution and schedule of all subtasks for all robots, which satisfy the constraint (2) in Section 2. Based on the permutation coding, four coding strategies have been developed in a previous study (Liu and Kroll, 2014) that focussed on the development and comparative analysis of these coding strategies. In this paper, one of these coding strategies, the task-based coding, will be selected to evaluate the effects of different mutation operators when solving multi-robot task allocation with cooperative tasks. This coding strategy was selected as it does not create infeasible solutions and as a genotype corresponds to just one phenotype. This avoids that the impact of the decoding strategy dilutes the performance comparison regarding mutation operators. In the following part, the encoding and the decoding of the task-based coding strategy are illustrated, respectively.

Encoding. Each gene represents a task. The genotype of a solution is composed of two parts:

- A *chromosome* is a string of genes and represents the sequence of all N^T tasks.
- A *gene-apportion* is a set of $N^R - 1$ integers, which splits a chromosome into N^R parts for N^R robots.

For example, Fig. 1(a) shows an example problem where six single-robot tasks and two cooperative tasks will be carried out by three robots. As each task is encoded as one gene, the genotype of a solution can be represented as shown in Fig. 1(b): the chromosome $\{1, 2, 3, 4, 5, 6, 7, 8\}$ is split into three segments by a gene-apportion $\{3, 6\}$ that is represented as two vertical lines.

T_l	P_i	T_l	P_i
T_1	P_1	T_6	P_6
T_2	P_2		P_9
T_3	P_3	T_7	P_7
T_4	P_4		P_8
T_5	P_5	T_8	P_{10}

(a) Tasks T_l and subtasks P_i

R_1			R_2			R_3	
1	2	3	4	5	6	7	8

(b) Genotype

Figure 1: An example with single-robot tasks ($T_1 - T_5, T_8$) and cooperative tasks (T_6, T_7)

Decoding. A genotype is decoded as a phenotype via two steps:

- (1) For single-robot tasks, each gene is directly decoded as its corresponding task; see Fig. 2(a).
- (2) For cooperative tasks, two subtasks of each cooperative task should be decoded. According to the genotype, it is obvious that each cooperative task is already assigned to a robot R_k , e.g. T_6 is assigned to $R_k = R_2$. Hence, the next step is to find the second robot so that two robots can carry it out cooperatively (satisfying the constraint EC1). For each cooperative task, the decoding is:
 - (S1) The closest subtask is assigned to robot R_k based on the traveling time c_{ijk}^t of robot R_k from one subtask P_i to another subtask P_j .
 - (S2) The other subtask is inserted at the “best” position of the task sequences of robots except R_k . The “best” position is the position that provides the least waiting time for performing this cooperative task,

which is calculated by enumerating all possible positions of the task sequences of robots except R_k . This decoding is carried out starting from the cooperative task that a robot meets first, so that all decoded phenotypes are feasible for execution.

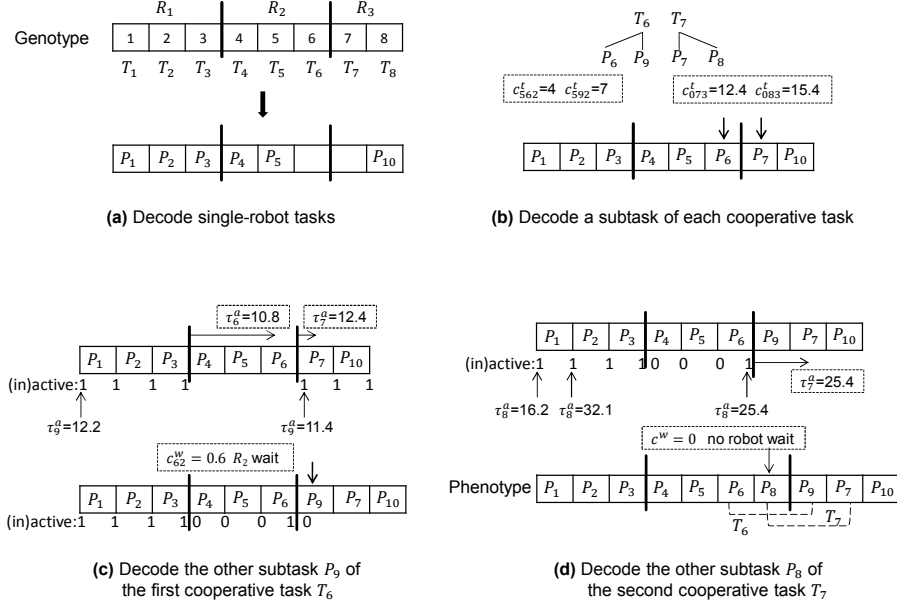


Figure 2: Decoding for a genotype in Fig. 1

For instance, the decoding outcome of the step “S1” is shown in Fig. 2(b): P_6 is assigned to R_2 because $c_{562}^t < c_{592}^t$; P_7 is assigned to R_3 after leaving its home base (denoted as “0”) because $c_{073}^t < c_{083}^t$. The decoding outcome of the step “S2” is shown in Fig. 2(c) and (d). Possible positions are marked as “active” (denoted as “1”), whereas impossible positions are marked as “inactive” (denoted as “0”). The decoding algorithm first finds the “best” position for P_9 because robot R_2 meets task T_6 earlier than robot R_3 meets task T_7 , i.e., $\tau_6^a < \tau_7^a$. There are seven possible positions for decoding P_9 except positions of robot R_2 , but only five positions will be tested: for R_1 , all four positions are tested; for R_3 , only the position before P_7 is tested because P_7 belongs to another cooperative task, which is performed in order

to satisfy the constraint EC3. As P_9 being inserted before P_7 provides the minimum waiting time, R_2 is waiting for $c_{62}^w = \tau_9^a - \tau_6^a = 0.6$ at P_6 until R_3 arrives at P_9 such that robots R_2 and R_3 can cooperatively perform T_6 (satisfying the constraints EC1 and EC2). In order to satisfy the constraint EC3, positions of chromosomes before either P_6 or before P_9 are marked as “inactive”; see Fig. 2(c). Therefore, only five active positions can be tested when decoding the next cooperative task P_8 (see Fig. 2(d)). Before assigning P_8 , the arriving time of R_3 at P_7 is recalculated, $\tau_7^a = 25.4$. The minimum waiting time $c^w = 0$ can be obtained when P_8 is inserted after P_6 . That is, R_2 arrives at P_8 and R_3 arrives at P_7 at the same time. The complete task allocation obtained using this decoding requires robots R_2 and R_3 as a coalition to execute T_6 and T_7 as shown in Fig. 2(d).

As illustrated above, this decoding can satisfy all executability constraints, i.e., all decoded phenotypes are feasible for execution.

Using this representation, each individual (solution candidate) includes a genotype and a phenotype. In the proposed genetic algorithm, the chromosomes of the genotypes are mutated for generating offspring; phenotypes are used to calculate the fitness.

3.2. Process of the developed genetic algorithm

The developed genetic algorithm in this paper is based on subpopulations. The main idea of the genetic algorithm is that selection and mutation are applied separately in each subpopulation. This genetic algorithm performed well when solving medium-scale traveling salesman problems (Liu and Kroll, 2012b). In this paper, the subpopulation-based genetic algorithm is used to solve multi-robot task allocation problems. The pseudo code of our proposed subpopulation-based genetic algorithm is presented in Algorithm 1.

Parameters. Parameters of the genetic algorithm are set at the beginning, such as population size (*pop_siz*), subpopulation size (*pop_sub*), elite count (*eli_cnt*), mutation probability (p_m), and termination criterion (*gen_num*).

Algorithm 1 Subpopulation-based genetic algorithm

```
1: Set parameters and select mutation operators
2: Generate an initial population
3: while termination criterion is not satisfied do
4:   for each genotype do
5:     Decode it and calculate the fitness value of its phenotype
6:   end for
7:   Divide the population into  $K$  non-overlapping subpopulations randomly
8:   for each subpopulation do
9:     Pass the eli_cnt superior individuals to the next generation directly
10:    Select the best_num superior individuals as parents
11:    for each parent do
12:       $n \leftarrow 0$ 
13:      repeat
14:         $n \leftarrow n + 1$ 
15:        Apply mutation operator to its chromosome with a probability  $p_m$ 
16:        Generate the new gene-apportion with a probability  $p_a$ 
17:      until  $n = (pop\_sub - eli\_cnt)/best\_num$ 
18:    end for
19:  end for
20:  Form the new population from all offspring in all subpopulations
21: end while
```

Initial population. The initial population is randomly produced based on the permutation coding, that is, both the chromosome and gene-apportion of each genotype in the initial population is generated at random.

Fitness calculation. All genotypes should be decoded as phenotypes according to the decoding procedure before fitness calculation. The fitness value of each individual is calculated according to the objective function (1).

New population. As can be seen from Algorithm 1, a new population is generated based on subpopulations. First, the whole population is randomly divided into non-overlapping subpopulations, and each subpopulation involves pop_sub individuals. After that, the elitism selection and mutation operators are applied to each subpopulation. The eli_cnt superior individuals are transferred to the new population, and the $best_num$ superior individuals are selected as parents. The $pop_sub - eli_cnt$ offspring are produced by mutating parents and generating new gene-apportions:

- The chromosome of a new offspring is produced by swap, insertion, inversion, or displacement mutation operators. *Swap* mutation exchanges two randomly selected genes. *Insertion* mutation moves a randomly chosen gene to another randomly chosen place. *Inversion* mutation reverses a randomly selected gene string. *Displacement* mutation inserts a random string of genes in another random place. Insertion can be considered as a special displacement.
- The gene-apportion of a new offspring is generated with a probability p_a ; otherwise, the gene-apportion of the parent is kept for the offspring. A gene-apportion is defined by $N^R - 1$ integers. Each element in a new gene-apportion is generated by rounding a number that is randomly selected within the range of $[1, N^T]$ according to a standard normal distribution (μ, σ^2) . μ is the cumulative average of the gene-apportion of the best individual obtained in each previous generation; $\sigma = 0.03N^T$ is used in this paper. This gene-apportion procedure will choose numbers, which are near to the cumulative average, with a higher probability.

Termination criterion. The genetic algorithm is terminated when the number of generations reaches a predefined number of generations (*gen_num*) in this paper. Both the population size and the number of generations are fixed in the simulation studies, i.e., the number of all produced individuals is constant. There are many alternative choices of the termination criterion, e.g. maximal number of generations, CPU time limit, and fitness limit/stall. In this paper, a fixed number of generations is used because (1) CPU time highly depends on the computer hardware, (2) what is a good fitness value is unpredictable, and (3) the convergence properties are uncertain.

3.3. Comparison of the subpopulation-based genetic algorithm and classical genetic algorithms

The main difference between the subpopulation-based genetic algorithm and classical genetic algorithms (Mitchell, 1998; Whitley, 2001) is the way of producing offspring. The parent selection of the proposed genetic algorithm can be considered as an extended tournament selection: superior individuals in each subpopulation are selected as parents. Hence, we compare the subpopulation-based genetic algorithm with classical genetic algorithms with tournament selection (see Algorithm 2). The number of parents in the proposed genetic algorithm is $best_num \cdot pop_siz / pop_sub$, while more parents $pop_siz - eli_cnt$ should be selected in classical genetic algorithms. Elites of classical genetic algorithms are chosen according to the fitness of all individuals in the whole population, while elites of the subpopulation-based genetic algorithm are selected according to the fitness of individuals in a subpopulation. Classical genetic algorithms can only keep *eli_cnt* best individuals, while the subpopulation-based genetic algorithm may keep local optima that may increase the population diversity.

Both crossover and mutation are used to produce offspring in classical genetic algorithms: crossover is applied with a high probability and mutation is applied with a small probability (Larrañaga et al., 1999; Kroll, 2013). Crossover dominates the search progress of classical genetic algorithms, which could be due to the fact that mutation trends to a random search. In the proposed genetic

Algorithm 2 A classical genetic algorithm with tournament selection

```
1: Generate an initial population
2: while termination criterion is not satisfied do
3:   for each genotype do
4:     Decode it and calculate the fitness value of its phenotype
5:   end for
6:   Pass the eli_cnt superior individuals to the next generation directly
7:   repeat
8:     Select tor_siz individuals randomly, the best of which is chosen as a parent
9:   until pop_sub – eli_cnt times
10:  Apply crossover to the chromosomes of each pair of parents with a probability
     $p_c$ 
11:  Apply mutation to the chromosome of each offspring obtained by crossover
    with a probability  $p_m$ 
12:  Generate the new gene-apportion for each offspring with a probability  $p_a$ 
13:  Form the new population (all offspring)
14: end while
```

algorithm, only mutation operators are performed with a probability of $p_m = 1$, which could be effective when combined with the proposed selection strategy. The *best_num* superior individuals in each subpopulation are mutated, while the rest is not used to produce offspring. The genetic algorithm exploits the solution space near to these superior individuals in this way. Non-overlapping subpopulations maintain local optima that keep diversity of the population and avoid premature convergence caused by a single superior individual. The performance of the subpopulation-based genetic algorithm will be analyzed in the next section.

The procedure of generating a new population in classical genetic algorithms is more complex than that in the proposed subpopulation-based genetic algorithm. The time complexity of the selection in classical genetic algorithms is $O(pop_siz - eli_cnt)$, because $pop_siz - eli_cnt$ parents are selected. As illustrated above, the time complexity of the selection in the subpopulation-based

genetic algorithm is $O(best_num \cdot pop_siz/pop_sub)$. The time complexity of swap is $O(1)$ as it is independent of the chromosome length. The time complexity of insertion, inversion, and displacement is $O(N^T)$ as in the worst case all genes have to be changed. Crossover is more complex than the above four mutation operators. Taking partially mapped crossover (PMX) (Larrañaga et al., 1999) as an example, the mapping relationship between selected *numg* genes from each pair of parents should be built to legalize the offspring. The time complexity of PMX is $O(numg + N^T)$ in the worst case: all *numg* genes should be mapped from one parent to the other and all genes have to be changed. As crossover is applied with a higher probability, classical genetic algorithms require more CPU time than the subpopulation-based genetic algorithm.

4. Simulation studies and analysis

In this section, the performance of the proposed genetic algorithm is analyzed when solving multi-robot task allocation problems without/with cooperative tasks. Four problems are tested in the simulation studies:

- *Prob.A* involves 90 single-robot tasks that are distributed in rows; its inspection area is similar to that shown in Fig. 3 but all tasks are single-robot tasks.
- *Prob.B* involves 100 single-robot tasks that are distributed in islands; its inspection area is similar to that shown in Fig. 4 but all tasks are single-robot tasks.
- *Prob.C* involves 80 single-robot tasks and 5 cooperative tasks, and all tasks are distributed in rows; see Fig. 3.
- *Prob.D* involves 90 single-robot tasks and 5 cooperative tasks, and all tasks are distributed in islands; see Fig. 4.

These scenarios have been used as test cases already in (Liu and Kroll, 2014; Liu, 2014) to compare the performance of different encoding and decoding strategies. Prob.A and Prob.B are multi-robot task allocation problems without

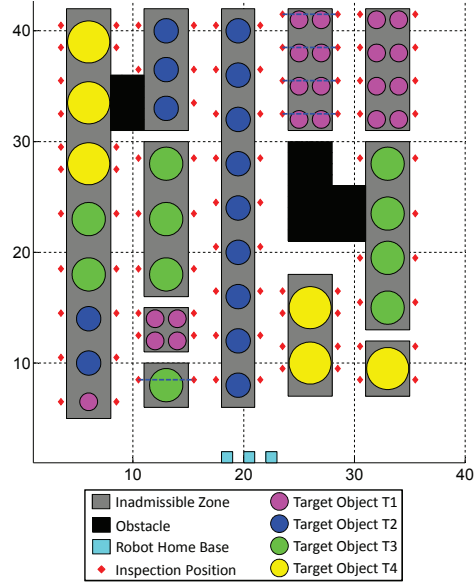


Figure 3: Inspection area and tasks of Prob.C (two subtasks of each cooperative task linked by a dashed line)

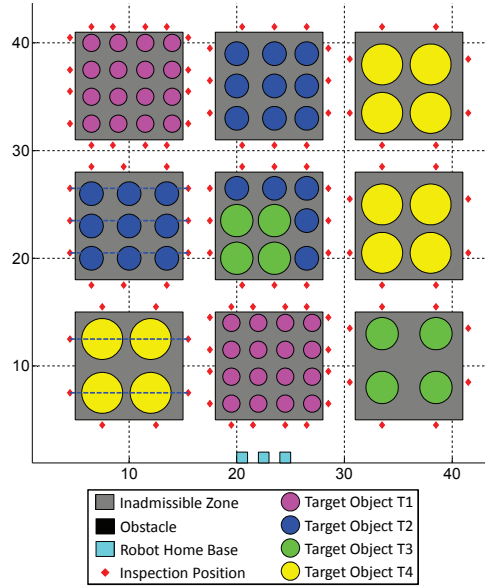


Figure 4: Inspection area and tasks of Prob.D (two subtasks of each cooperative task linked by a dashed line)

cooperative tasks. Prob.C and Prob.D are multi-robot task allocation problems with cooperative tasks.

In the experiments, each tested genetic algorithm is performed with a population size of $pop_siz = 200$ and the number of generations chosen as $gen_num = 10^4$. To statistically evaluate the performance of the proposed genetic algorithm, 20 independent runs of each algorithm are implemented on an Intel Core i3 PC with 3.2 GHz, 8 GB (RAM), Windows 7 Professional, MATLAB R2011b. More runs could provide more accurate results but require more CPU time. Hence, 20 independent runs are carried out to restrict the computational effort, and analysis of variance (ANOVA) is used to check whether the performance differences (solution quality) between the different genetic algorithms are statistically significant. If the value of the significance level is smaller than 0.05, the effects of genetic algorithms are assessed to be statistically significant at a level of confidence of 95%.

4.1. Case study 1: Subpopulation-based vs. binary tournament GA

The first case study compares the performance of the subpopulation-based genetic algorithm with a classical genetic algorithm with binary tournament selection. The frameworks of both genetic algorithms are displayed in Algorithm 1 and Algorithm 2 in Section 3, and the parameters of two genetic algorithms are listed in Table 1. Inversion mutation is used in both genetic algorithms because it performs better than other mutation operators when solving combinatorial optimization problems without cooperative tasks (Wang and Zhang, 2006; Albayrak and Allahverdi, 2011; Deep and Mebrahtu, 2011; Liu and Kroll, 2012b).

The experimental results are recorded in Table 2, which indicate that the proposed subpopulation-based genetic algorithm provides better solutions and requires less CPU time than the classical genetic algorithm. An ANOVA test shows that the differences of the solution quality between these two genetic algorithms are statistically significant. Randomly choosing 5 from the 20 runs of each genetic algorithm, the solution quality (completion time) of the best

Table 1: Parameter choice in the experiments

Parameter	Subpopulation-based GA	Classical GA
<i>pop_sub</i>	10	–
<i>tor_siz</i>	–	2
<i>eli_cnt</i>	2	2
<i>best_num</i>	1	–
p_c	–	0.9
p_m	1	0.01
p_a	0.2	0.2
Crossover	–	PMX
Mutation	Inversion	Inversion

Table 2: Completion time J in sec. and average CPU time in sec. for different genetic algorithms

Problem	Criterion	Subpopulation-based GA	Classical GA
Prob.A	J_{\min}	170.06	250.03
	J_{mean}	189.55	290.07
	J_{\max}	225.56	319.12
	CPU	988	1432
Prob.B	J_{\min}	185.95	257.16
	J_{mean}	207.03	300.45
	J_{\max}	228.75	355.11
	CPU	1028	1423
Prob.C	J_{\min}	252.72	348.52
	J_{mean}	292.78	414.79
	J_{\max}	376.46	500.94
	CPU	2419	2732
Prob.D	J_{\min}	255.96	374.93
	J_{mean}	333.07	448.25
	J_{\max}	383.95	480.42
	CPU	2580	2885

J_{\max} – Maximum completion time; J_{mean} – Mean completion time; J_{\min} – Minimum completion time.

solution candidate in each generation is shown in Fig. 5. It is obvious that the subpopulation-based genetic algorithm converges significantly faster than the classical genetic algorithm within the first 1000 generations.

This case study indicates that the proposed genetic algorithm based on

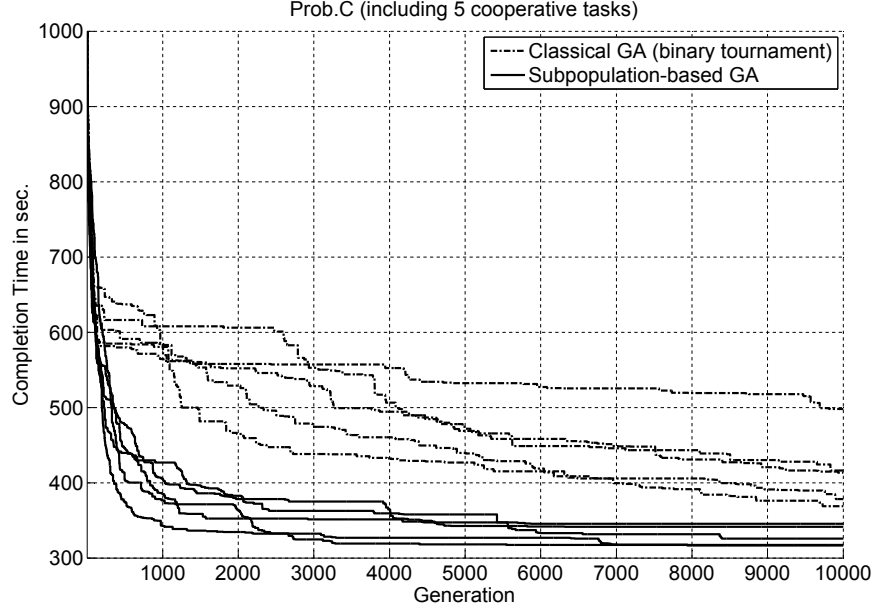


Figure 5: The search progress of two genetic algorithms for solving Prob.C (5 runs selected from the total 20 runs of each algorithm)

subpopulations perform better than the classical genetic algorithm with PMX crossover and tournament selection when solving multi-robot task allocation problems, especially when requiring less CPU time and less generations. The subpopulation-based genetic algorithm can also provide significantly better solutions than the following two classical genetic algorithms: (1) a classical genetic algorithm with a large tournament size $tor_siz = 10$; (2) a classical genetic algorithm without PMX crossover and with only inversion mutation ($p_m = 1$). The experimental results of solving Prob.A and Prob.C are shown in Fig. 6, indicating that the genetic algorithm only employing mutation must be combined with an effective selection, so that the algorithm can perform well.

As discussed in Section 3, the proposed genetic algorithm selects parents based on subpopulations that performs well in conjunction with mutation operators. Performing the selection of the elites in each subpopulation can enhance the exploration in the search space, because it keeps both the global and local

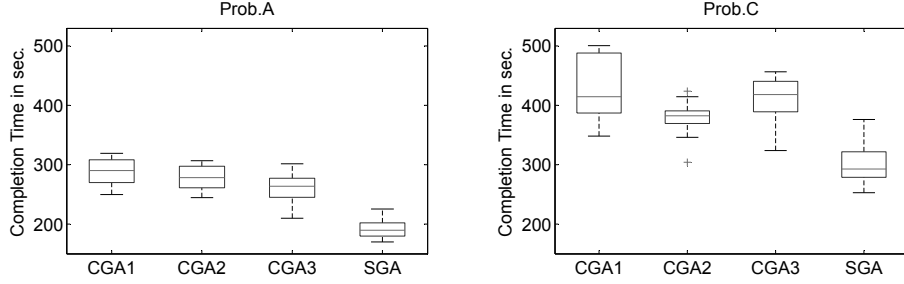


Figure 6: The distribution of the solution quality of different genetic algorithms for solving Prob.A and Prob.C (20 runs): SGA – subpopulation-based GA in Table 1; CGA1/CGA2/CGA3 – classical GA in Table 1 but GA2 with *tor_siz* = 10, GA3 with $p_c = 0$ and $p_m = 1$

optimal solutions that avoid the algorithm being dominated by a single superior individual. Only the best individual in each subpopulation is chosen as the parent of mutation. In this way, the search space near the parents can be well exploited.

We also test the subpopulation-based genetic algorithm with different probabilities for generating new gene-appointions $p_a = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$, and the results in Table 3 show that the differences of the solution quality are not statistically significant when solving all investigated problems. It is obvious that a large p_a results in more CPU time. Therefore, $p_a = 0.2$ is chosen in the following experiments for analyzing the mutation effects of the subpopulation-based genetic algorithm. The subpopulation-based genetic algorithm with a different number of parents $best_num = \{1, 2, 4\}$ is performed to solve all investigated problems; see Table 4. The results of solving Prob.A and Prob.B show that the proposed algorithm with $best_num = \{1, 2\}$ performs better than with $best_num = 4$. However, the differences between $best_num = \{1, 2, 4\}$ are not statistically significant when solving Prob.C and Prob.D. In the following case studies, $best_num = 1$ is chosen.

Table 3: Average completion time J_{mean} in sec. for the subpopulation-based genetic algorithm with different p_a

p_a	0	0.2	0.4	0.6	0.8	1	Sig.Level*
Prob.A	192.78	189.55	196.62	200.20	200.89	196.03	0.25
Prob.B	207.98	207.03	205.02	204.52	209.12	212.85	0.25
Prob.C	294.18	292.78	299.76	298.32	305.86	290.78	0.95
Prob.D	299.03	308.42	306.71	300.72	301.82	295.35	0.63

* The significance level obtained by an ANOVA test. If the value of Sig.Level is smaller than 0.05, the differences of solution quality between these p_a are statistically significant.

Table 4: Average completion time J_{mean} in sec. for the subpopulation-based genetic algorithm with different $best_num$

$best_num$	1	2	4	Sig.Level*
Prob.A	189.55	188.92	204.96	0.00
Prob.B	207.03	210.83	216.94	0.02
Prob.C	292.78	310.35	305.20	0.76
Prob.D	333.07	340.78	328.4872	0.54

* The significance level obtained by an ANOVA test. If the value of Sig.Level is smaller than 0.05, the differences of solution quality between these $best_num$ are statistically significant.

4.2. Case study 2: Subpopulation-based GA with single mutation operator

The second and the third case studies analyze the effects of the subpopulation-based genetic algorithm with different mutation operators and their combinations. Swap, insertion, inversion, and displacement mutation operators are investigated in this paper. The tested subpopulation-based genetic algorithms are listed in Table 5.

This case study tests the performance of the subpopulation-based genetic algorithms with a single mutation operator (GA1–GA4 in Table 5); each mutation operator produces $pop_sub - eli_cnt = 8$ offspring in each subpopulation. The results are shown in Fig. 7. An ANOVA test shows that: (1) inversion (GA3) performs significantly better than the other three mutation operators when solving Prob.A and Prob.B; (2) the differences of the solution quality are not statistically significant when using swap, inversion, and displacement to solve Prob.C and Prob.D.

Table 5: Subpopulation-based genetic algorithm with different mutation operators

Genetic algorithm	Mutation operator(s)
GA1	Swap
GA2	Insertion
GA3	Inversion
GA4	Displacement
GA5	Swap and inversion
GA6	Insertion and inversion
GA7	Displacement and inversion
GA8	Swap, insertion, inversion, and displacement

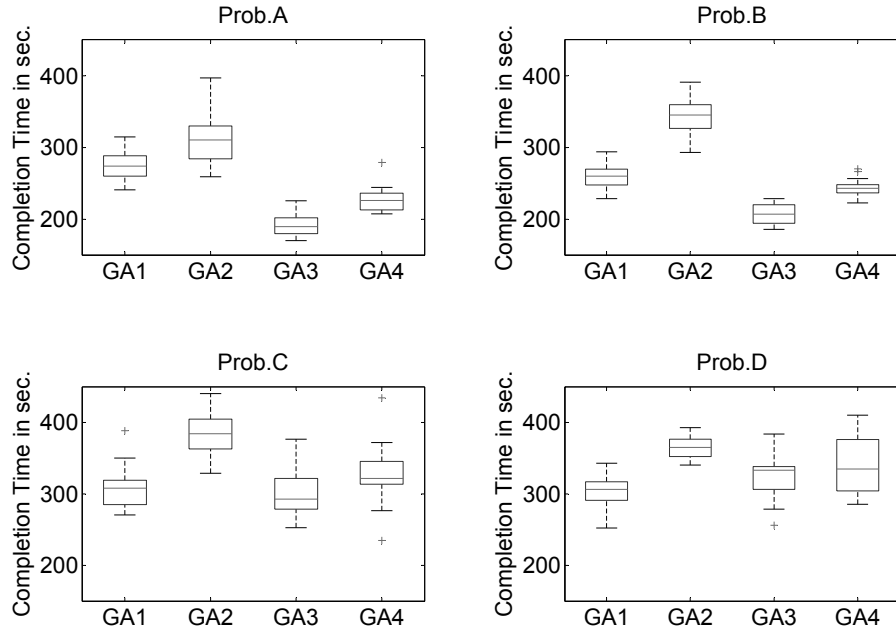


Figure 7: The distribution of the solution quality of the subpopulation-based genetic algorithms with a single mutation operator (20 runs)

4.3. Case study 3: Subpopulation-based GA with multiple mutation operators

The third case study analyzes the performance of the subpopulation-based genetic algorithms with multiple mutation operators (GA5–GA8 in Table 5). Each mutation operator in GA5–GA7 produces 4 offspring in each subpop-

ulation by repeated application; each mutation operator in GA8 produces 2 offspring in each subpopulation. Inversion is combined with the other mutation operators in this case study, because it performed well in the second case study. The experimental results is displayed in Fig. 8. An ANOVA test shows that: (1) the differences of the solution quality between GA5–GA8 are not statistically significant when solving Prob.A and Prob.B; (2) GA5 and GA8 can provide significantly better solutions than GA6 and GA7 when solving Prob.C and Prob.D.

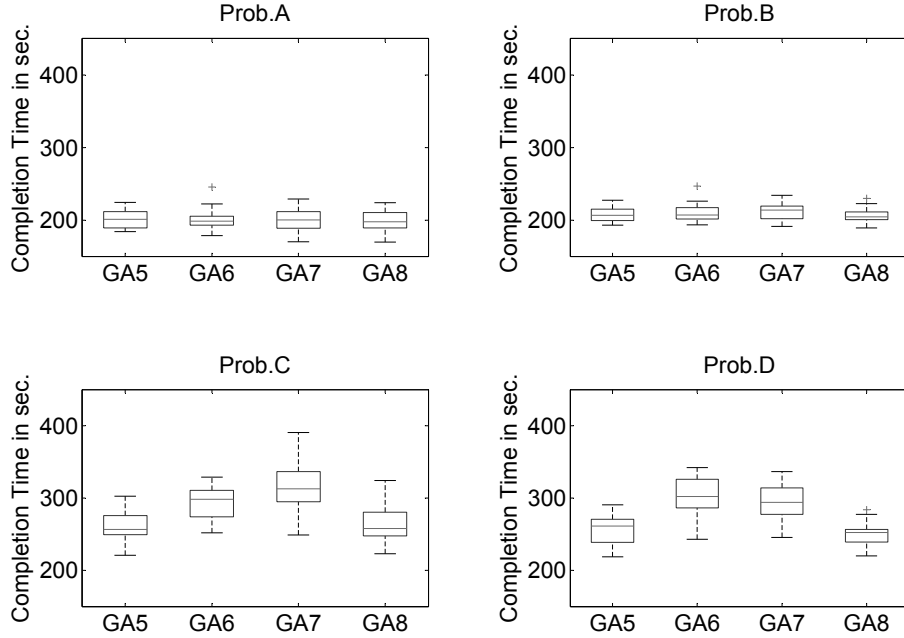


Figure 8: The distribution of the solution quality of the subpopulation-based genetic algorithm with multiple mutation operators (20 runs)

The results of all tested subpopulation-based genetic algorithms listed in Table 5 are shown in Fig. 9 and Table 6. GA3, GA5, and GA8 can provide better solutions than the other genetic algorithms. An ANOVA test shows that: (1) the differences of the solution quality using GA3, GA5, GA6, GA7, and GA8 are not statistically significant when solving Prob.A and Prob.B; (2) GA5 and GA8 perform significantly better than the other tested genetic algorithms when

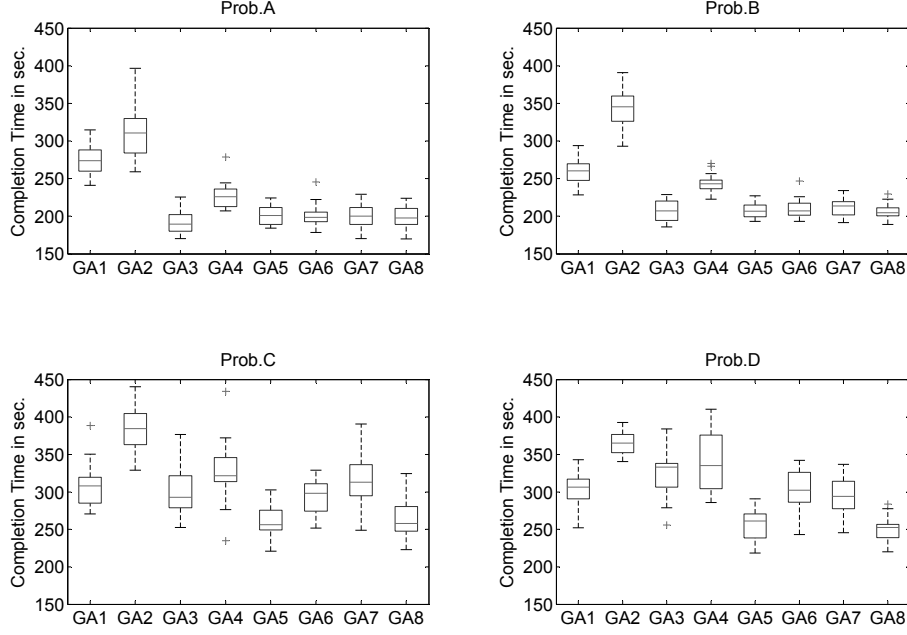


Figure 9: The distribution of the solution quality of the subpopulation-based genetic algorithm with different mutation operators (20 runs)

solving Prob.C and Prob.D.

4.4. Discussion

In general, it is difficult to find the *best* mutation operator that could produce all desired effects. The influences of mutation operators vary in different genetic algorithms and in solving different problems. As illustrated above, inversion performs well to solve multi-robot task allocation problems without cooperative tasks, which is similar to the study of solving traveling salesman problems (Deep and Mebrahtu, 2011; Albayrak and Allahverdi, 2011; Liu and Kroll, 2012b). The swap and inversion combination performs well to solve multi-robot task allocation problems with cooperative tasks. Swap and inversion are efficient in the simulation studies, which could be due to the fact that they can improve solution candidates with crossed paths effectively.

In general, good solutions do not include crossed paths or only include a few

Table 6: Completion time J in sec. for the subpopulation-based genetic algorithm with different mutation operators (best results highlighted in bold face)

Problem	Criterion	GA1	GA2	GA3	GA4	GA5	GA6	GA7	GA8
Prob.A	J_{\min}	240.87	259.00	170.06	207.28	184.07	178.41	170.10	169.73
	J_{mean}	273.98	310.50	189.55	225.93	200.86	198.31	200.00	197.73
	J_{\max}	314.81	396.67	225.56	278.89	224.40	245.39	229.00	223.99
Prob.B	J_{\min}	228.42	293.00	185.95	222.65	193.22	193.25	191.36	189.00
	J_{mean}	260.16	345.23	207.03	243.05	206.74	206.91	213.62	204.46
	J_{\max}	294.05	390.84	228.75	270.26	227.21	246.90	234.23	229.72
Prob.C	J_{\min}	270.50	328.82	252.72	234.96	220.76	251.82	248.93	222.82
	J_{mean}	308.14	384.37	292.78	321.59	256.44	298.16	312.96	257.80
	J_{\max}	388.62	440.42	376.46	434.31	302.74	328.96	390.78	324.31
Prob.D	J_{\min}	252.29	340.35	255.96	285.73	218.58	242.96	245.62	220.00
	J_{mean}	306.57	365.23	333.07	335.02	261.11	302.09	294.19	252.44
	J_{\max}	343.06	392.85	383.95	410.17	290.83	342.09	336.63	283.71

¹ GA1–Swap; GA2–Insertion; GA3–Inversion; GA4–Displacement; GA5–Swap and inversion; GA6–Insertion and inversion; GA7–Displacement and inversion; GA8–Swap, insertion, inversion, and displacement.

² Prob.A and Prob.B without cooperative tasks; Prob.C and Prob.D with cooperative tasks.

crossed paths. Fig. 10(a) shows an example where one cross may occur. This allocation can be improved by inverting $\{5, 4, 3, 2\}$; see Fig. 10(b). Fig. 11(a) shows another example where two crosses may occur. This allocation can be improved by swapping $\{1\}$ and $\{6\}$; see Fig. 11(b). If applying inversion to this allocation, the inversion mutation has to be used two times appropriately; see Fig. 11(c). These two examples imply that proper swap is more efficient than inversion in case of many crossed paths. Insertion and displacement could not effectively improve these allocations. On the contrary, inappropriate swap produces worse solutions than inversion, e.g. swap produces two crosses, while inversion produces one cross in Fig. 12. Therefore, inversion can obtain better results than swap if given a large number of generations.

The mentioned improvement is observable from Fig. 10 and Fig. 11 when solving problems where the cost of one robot does not influence the costs of the other robots such as multi-robot task allocation problems without cooperative tasks. It becomes more complex in case of problems with cooperative tasks as the cost of finishing cooperative tasks does not depend on the task allocation

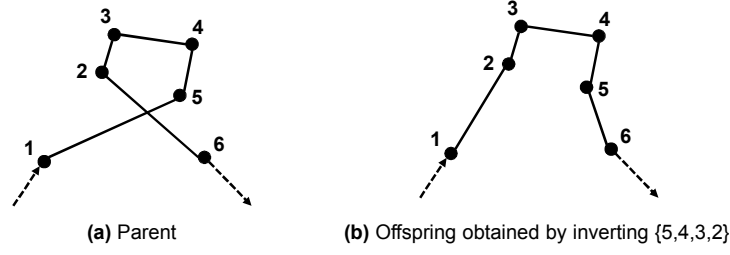


Figure 10: An example with one cross for inversion

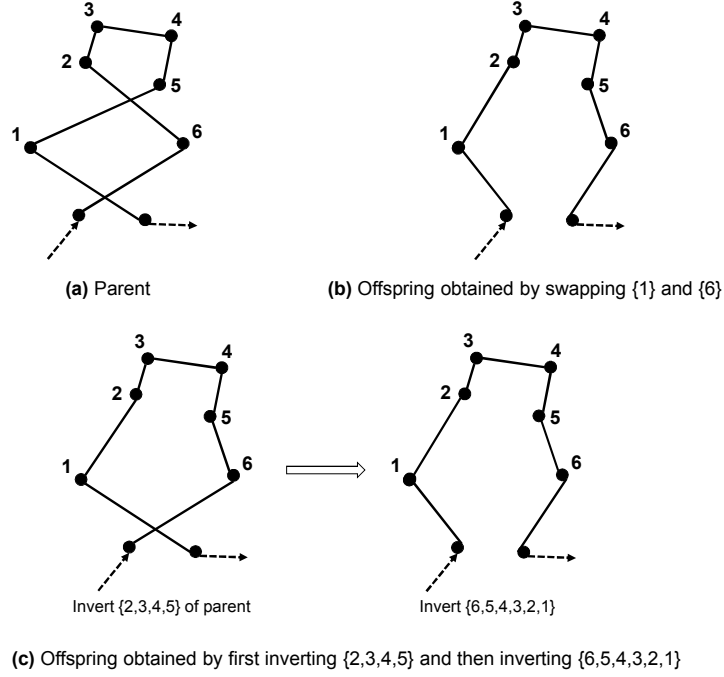


Figure 11: An example with two crosses for swap and inversion

for one robot but for two robots. Swap and inversion in Fig. 10 and Fig. 11 may cause a longer waiting time and a longer completion time.

To statistically evaluate the effect of different generations gen_num , the distribution (20 runs) of the solution quality of eight subpopulation-based genetic algorithms in different generations is analyzed. The experimental results of solving problems without cooperative tasks (Prob.A and Prob.B) are similar: swap

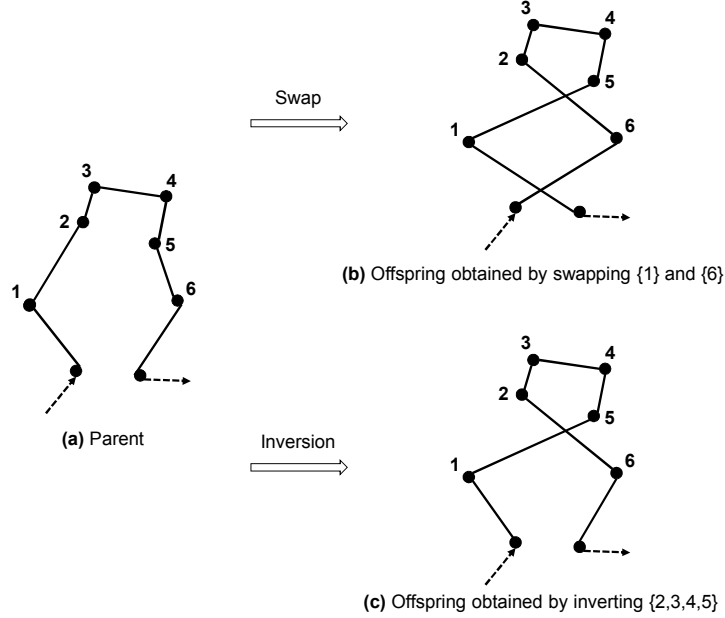


Figure 12: An example of inappropriate swap and inversion

(GA1) obtains better solutions than inversion (GA3) within 100 generations, while inversion produces better solutions than swap after 100 generations; see Fig. 13. In this part, we focus on the performance of genetic algorithms when solving problems with cooperative tasks (Prob.C and Prob.D). Similar results can be obtained when solving these two problems, e.g. Fig. 14. The results indicate that swap (GA1) obtains better solutions than inversion (GA3) within 500 generations, while the swap-inversion combination (GA5) produces better solutions than swap after 500 generations. Many crossed paths may occur in the early generations due to the randomly generated initial population and a small number of generations. Hence, swap is more efficient than inversion in the early generations. After 500 generations, the differences of the solution quality using swap and inversion are not statistically significant. The swap-inversion combination performs well because swap and inversion are applied to the same parent in each subpopulation. In this case, the good capabilities of both mutation operators are preserved. As discussed before, multiple mutation operators

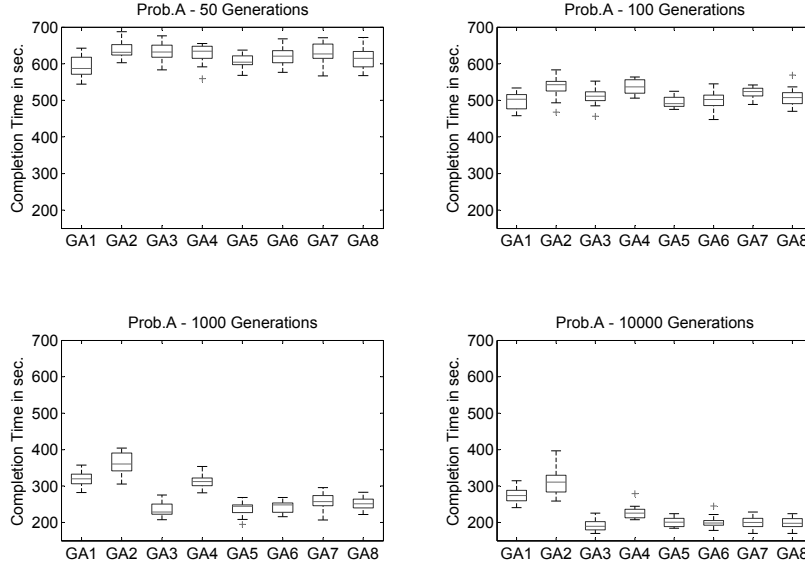


Figure 13: The distribution of the solution quality of eight subpopulation-based genetic algorithms in different generations for solving Prob.A (20 runs)

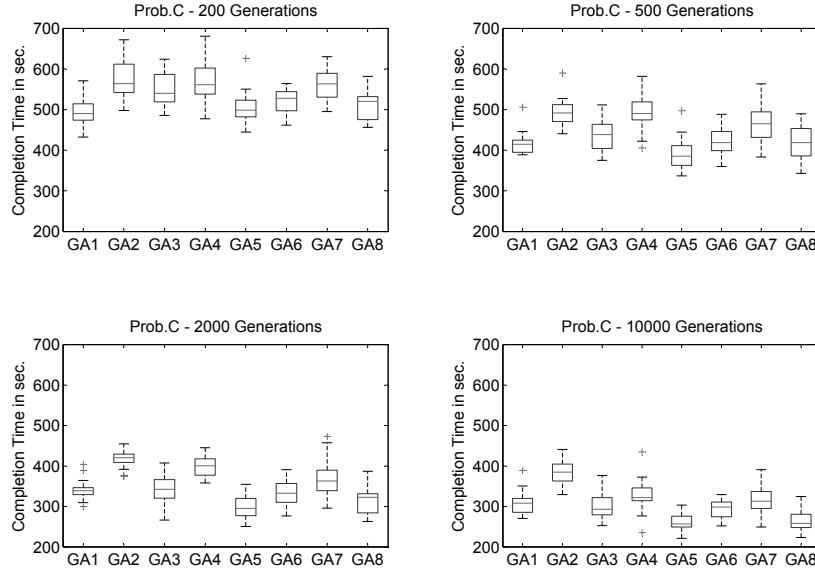


Figure 14: The distribution of the solution quality of eight subpopulation-based genetic algorithms in different generations for solving Prob.C (20 runs)

involving swap and inversion are suggested to solve multi-robot task allocation problems, especially with cooperative tasks.

5. Conclusion and outlook

The problem complexity significantly increases if cooperative tasks are involved because they introduce additional spatial and temporal constraints. In this paper, the performance of different mutation operators in a subpopulation-based genetic algorithm is analyzed for solving multi-robot task allocation problems without/with cooperative tasks. So far, a little work addresses this problem area. The proposed subpopulation-based genetic algorithm uses just inversion mutation and selection, though obtains better solutions than classical genetic algorithms with tournament selection, partially mapped crossover (PMX), and inversion mutation in the test cases. Succeeding, a subpopulation-based genetic algorithm with four alternative mutation operators or with four mutation operator combinations was tested to find suitable mutation operators for multi-robot task allocation problems. The results indicate that inversion mutation performs well when solving problems without cooperative tasks, and a swap-inversion combination performs well when solving problems with cooperative tasks. As it is difficult to produce all desired effects with a single mutation operator, using multiple mutation operators is suggested, especially to solve complex problems.

The rate of each mutation operator is constant in this paper. As the performance of different mutation operators varies in different generations, future work will focus on employing an adaptive rate of mutation operators to improve the performance of the genetic algorithm. Future work will also include more test scenarios, especially problems with a larger number of tasks. The problem complexity increases with the number of robots required for each cooperative task. In this paper, each cooperative task requires only two robots to carry it out simultaneously. In future work, the performance of the genetic algorithm will be analyzed when solving problems with cooperative tasks that require more than two robots to execute cooperatively.

Acknowledgements

This work was supported by the scholarship awarded by the China Scholarship Council (CSC) and the Completion Scholarship awarded by the University of Kassel (Abschlussstipendien für Promovierende der Universität Kassel), which are greatly acknowledged.

Conflict of interest

The authors declare that they have no conflict of interest.

References

References

- Akpınar, S., Bayhan, G.M., 2011. A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. *Engineering Applications of Artificial Intelligence* 24, 449–457. doi:10.1016/j.engappai.2010.08.006.
- Albayrak, M., Allahverdi, N., 2011. Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems With Applications* 38, 1313–1320. doi:10.1016/j.eswa.2010.07.006.
- Bonow, G., Kroll, A., 2013. Gas leak localization in industrial environments using a TDLAS-based remote gas sensor and autonomous mobile robot with the Tri-Max method, in: *IEEE International Conference on Robotics and Automation (ICRA 2013)*, Piscataway, NJ: IEEE Press, Karlsruhe, Germany. pp. 987–992.
- Brizuela, C.A., Aceves, R., 2003. Experimental genetic operators analysis for the multi-objective permutation flowshop, in: Fonseca, C., Fleming, P., Zitzler, E., Thiele, L., Deb, K. (Eds.), *Evolutionary Multi-Criterion Optimization*. Springer Berlin Heidelberg. volume 2632 of *Lecture Notes in Computer Science*, pp. 578–592. doi:10.1007/3-540-36970-8_41.

- Deb, D., Deb, K., 2012. Investigation of mutation schemes in real-parameter genetic algorithms, in: Panigrahi, B., Das, S., Suganthan, P., Nanda, P. (Eds.), *Swarm, Evolutionary, and Memetic Computing*. Springer Berlin Heidelberg. volume 7677 of *Lecture Notes in Computer Science*, pp. 1–8. doi:10.1007/978-3-642-35380-2_1.
- Deep, K., Mebrahtu, H., 2011. Combined mutation operators of genetic algorithm for the travelling salesman problem. *International Journal of Combinatorial Optimization Problems and Informatics* 2, 1–23.
- Deep, K., Thakur, M., 2007. A new mutation operator for real coded genetic algorithms. *Applied Mathematics and Computation* 193, 211–230. doi:10.1016/j.amc.2007.03.046.
- Fogel, D.B., Atmar, J., 1990. Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics* 63, 111–114.
- Gerkey, B.P., Matarić, M.J., 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research* 23, 939–954.
- Hasan, B.H.F., Saleh, M.S.M., 2011. Evaluating the effectiveness of mutation operators on the behavior of genetic algorithms applied to non-deterministic polynomial problems. *Informatica* 35, 513–518.
- Holland, J.H., 1992. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA.
- Hong, T.P., Wang, H.S., Chen, W.C., 2000. Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics* 6, 439–455. doi:10.1023/A:1009642825198.
- Karthikeyan, P., Baskar, S., Alphones, A., 2013. Improved genetic algorithm using different genetic operator combinations (GOCs) for mul-

- ticast routing in ad hoc networks. *Soft Computing* 17, 1563–1572. doi:10.1007/s00500-012-0976-4.
- Kociecki, M., Adeli, H., 2014. Two-phase genetic algorithm for topology optimization of free-form steel space-frame roof structures with complex curvatures. *Engineering Applications of Artificial Intelligence* 32, 218–227. doi:10.1016/j.engappai.2014.01.010.
- Kroll, A., 2013. *Computational Intelligence – Eine Einführung in Probleme, Methoden und technische Anwendungen*. Oldenbourg Verlag, München, Germany.
- Kwak, N.S., Lee, J., 2011. An implementation of new selection strategies in a genetic algorithm – population recombination and elitist refinement. *Engineering Optimization* 43, 1367–1384. doi:10.1080/0305215X.2011.558577.
- Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S., 1999. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review* 13, 129–170. doi:10.1023/A:1006529012972.
- Liu, C., 2014. *Multi-Robot Task Allocation for Inspection Problems with Cooperative Tasks Using Hybrid Genetic Algorithms*. Ph.D. thesis. Department of Measurement and Control, Mechanical Engineering, University of Kassel. URL: <http://nbn-resolving.de/urn:nbn:de:hebis:34-2014101646126>.
- Liu, C., Kroll, A., 2012a. A centralized multi-robot task allocation for industrial plant inspection by using A* and genetic algorithms, in: 11th International Conference on Artificial Intelligence and Soft Computing (ICAISA 2012), Heidelberg, Dordrecht, London, New York: Springer, Zakopane, Poland. pp. 466–474.
- Liu, C., Kroll, A., 2012b. On designing genetic algorithms for solving small- and medium-scale traveling salesman problems, in: International Symposium

- on Swarm Intelligence and Differential Evolution (SIDE 2012), Heidelberg, Dordrecht, London, New York: Springer, Zakopane, Poland. pp. 283–291.
- Liu, C., Kroll, A., 2014. Memetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks. *Soft Computing* doi:10.1007/s00500-014-1274-0.
- Mc Ginley, B., Maher, J., O’Riordan, C., Morgan, F., 2011. Maintaining healthy population diversity using adaptive crossover, mutation, and selection. *Ieee Transactions On Evolutionary Computation* 15, 692–714. doi:10.1109/TEVC.2010.2046173.
- Mitchell, M., 1998. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA.
- Ordoñez Müller, A., Kroll, A., 2013. Effects of beam divergence in hand-held TDLAS sensors on long distance gas concentration measurements, in: *International Workshop on Advanced Infrared Technology and Applications (AITA 2013)*, Turin, Italy. pp. 9–13.
- Ordoñez Müller, A., Kroll, A., 2014. On the use of cooperative autonomous mobile robots and optical remote sensing in inspection robotics, in: *Automation 2014*, Baden-Baden, Germany. pp. 847–864.
- Nearchou, A.C., 2004. The effect of various operators on the genetic search for large scheduling problems. *International Journal of Production Economics* 88, 191–203. doi:10.1016/S0925-5273(03)00184-1.
- Osaba, E., Carballido, R., Diaz, F., Onieva, E., de la Iglesia, I., Perallos, A., 2014. Crossover versus mutation: a comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems. *The Scientific World Journal* doi:10.1155/2014/154676.
- Serpell, M., Smith, J.E., 2010. Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation* 18, 491–514. doi:10.1162/EVC0_a_00006.

- Spears, W.M., 1992. Crossover or mutation?, in: Proceedings of the Second Workshop on Foundations of Genetic Algorithms, Morgan Kaufmann, Vail, Colorado, USA. pp. 221–237.
- Taplin, J.H.E., Qui, M., Salim, V.K., Han, R., 2005. Cost-benefit Analysis and Evolutionary Computing: Optimal Scheduling of Interactive Road Projects. Edward Elgar Publishing, Massachusetts, USA.
- Walkenhorst, J., Bertram, T., 2011. Multikriterielle optimierungsverfahren für pickup-and-delivery-probleme, in: Proceedings of 21. Workshop Computational Intelligence, Dortmund, Germany. pp. 61–76.
- Wang, L., Zhang, L., 2006. Determining optimal combination of genetic operators for flow shop scheduling. The International Journal of Advanced Manufacturing Technology 30, 302–308. doi:10.1007/s00170-005-0082-1.
- Whitley, D., 2001. An overview of evolutionary algorithms: practical issues and common pitfalls. Information and Software Technology 43, 817 – 831. doi:10.1016/S0950-5849(01)00188-4.
- Yuan, S., Skinner, B., Huang, S., Liu, D., 2013. A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms. European Journal of Operational Research 228, 72–82. doi:10.1016/j.ejor.2013.01.043.